Date: 18 - 01 - 2026
Assignment No.: 2

Name: Varghese K James
Reg No.: 3122245001192

Excercise 2 - Implement Informed Search Strategy - A* Searach

Aim: Implement Informed Search Strategy - A* Searach for Emergency Evacuation Planning in a Building Grid

Source Code

```
import heapq

class Search_problem:
    def start_node(self):
        pass
    def is_goal(self, node):
        pass
    def neighbors(self, node):
        pass
    def heuristic(self, node):
        return 0

class Edge:
    def __init__(self, from_node, to_node, action=None, cost=1):
        self.from_node = from_node
        self.to_node = to_node
        self.action = action
        self.cost = cost
        if self.cost < 0:
            raise ValueError("Edge cost must be non-negative.")

    def __repr__(self):
        return f"Edge({self.from_node} -> {self.to_node}, cost={self.cost})"

class GridSearchProblem(Search_problem):
    def __init__(self, grid_str):
        self.grid = [list(line) for line in grid_str.strip().split('\n')]
        self.rows = len(self.grid)
        self.cols = len(self.grid[0])
        self.start = None
        self.exits = []

        # To find Start and Exits
        for r in range(self.rows):
            for c in range(self.cols):
                if self.grid[r][c] == 'S':
                    self.start = (r, c)
                elif self.grid[r][c] == 'E':
                    self.exits.append((r, c))
```

```
    def start_node(self):
        return self.start

    def is_goal(self, node):
        return node in self.exits

    def neighbors(self, node):
        r, c = node
        moves = [(-1, 0, 'Up'), (1, 0, 'Down'), (0, -1, 'Left'), (0, 1,
'Right')]
        result = []

        for dr, dc, action in moves:
            nr, nc = r + dr, c + dc
            if 0 <= nr < self.rows and 0 <= nc < self.cols:
                if self.grid[nr][nc] != '#':
                    result.append(Edge(node, (nr, nc), action, cost=1))
        return result

    def heuristic(self, node):
        if not self.exits:
            return 0
        r, c = node
        distances = [abs(r - er) + abs(c - ec) for er, ec in self.exits]
        return min(distances)

class Path:
    def __init__(self, start_node=None, parent_path=None, edge=None):
        if parent_path is None:
            self.last_node = start_node
            self.parent_path = None
            self.cost = 0 # g(n)
        else:
            self.last_node = edge.to_node
            self.parent_path = parent_path
            self.cost = parent_path.cost + edge.cost

    def end(self):
        return self.last_node

    def __repr__(self):
        nodes = []
        current = self
        while current:
            nodes.append(str(current.last_node))
            current = current.parent_path
        nodes.reverse()
        return f"Path(len={self.cost}, nodes={nodes})"

class Searcher:
```

```
    def __init__(self, problem):
        self.problem = problem
        self.nodes_expanded = 0

    def search(self):
        print("Starting A* Search...")
        start = self.problem.start_node()
        initial_path = Path(start_node=start)

        frontier = []
        tie_breaker_index = 0

        # f(n) = g(n) + h(n)
        initial_h = self.problem.heuristic(start)
        initial_f = initial_path.cost + initial_h

        heapq.heappush(frontier, (initial_f, tie_breaker_index, initial_path))
        tie_breaker_index += 1

        explored = set()

        while frontier:
            f_val, _, current_path = heapq.heappop(frontier)
            current_node = current_path.end()

            if current_node in explored:
                continue

            explored.add(current_node)
            self.nodes_expanded += 1

            if self.problem.is_goal(current_node):
                print(f"Goal met! Nodes expanded: {self.nodes_expanded}")
                return current_path

            neighbors = self.problem.neighbors(current_node)

            for edge in neighbors:
                child_node = edge.to_node
                if child_node not in explored:
                    new_path = Path(parent_path=current_path, edge=edge)

                    g_score = new_path.cost
                    h_score = self.problem.heuristic(child_node)
                    f_score = g_score + h_score

                    heapq.heappush(frontier, (f_score, tie_breaker_index,
new_path))
                    tie_breaker_index += 1
```

```
        return None

if __name__ == "__main__":
    grid1_str = """
S . . . . . .
# # # # # # .
. . . . . # .
. # # # . # .
. . . # . # E
. # . # . . .
. # . . . . .
"""
    print("Test Case: GRID 1")
    problem = GridSearchProblem(grid1_str)
    searcher = Searcher(problem)
    result = searcher.search()
    if result:
        print(f"Final Path Cost: {result.cost}")
        print(f"Final Path: {result}")
    else:
        print("No path found")

    print()


    grid2_str = """
S . . . . . . . .
# # # # # . # # #
. . . . # . . . #
. # # . # # # . #
. # . . . . . . .
. # . # # # # # .
. . . . . . . . .
# # # # # . # # #
. . . . . . . . E
"""
    print("Test Case: GRID 2")
    problem = GridSearchProblem(grid2_str)
    searcher = Searcher(problem)
    result = searcher.search()
    if result:
        print(f"Final Path Cost: {result.cost}")
        print(f"Final Path: {result}")
    else:
        print("No path found")
```

Input/Output

Name: Varghese K James
Reg No.: 3122245001192

```
vicfic@NAG3:~/stud/ai/ex2$ python a-star.py
Test Case: GRID 1
Starting A* Search...
Goal met! Nodes expanded: 49
Final Path Cost: 16
Final Path: Path(len=16, nodes=['(0, 0)', '(0, 1)', '(0, 2)', '(0, 3)', '(0, 4)', '(0,
 5)', '(0, 6)', '(0, 7)', '(0, 8)', '(0, 9)', '(0, 10)', '(0, 11)', '(1, 11)', '(2, 11
)', '(3, 11)', '(4, 11)', '(4, 12)'])

Test Case: GRID 2
Starting A* Search...
Goal met! Nodes expanded: 116
Final Path Cost: 24
Final Path: Path(len=24, nodes=['(0, 0)', '(0, 1)', '(1, 1)', '(2, 1)', '(3, 1)', '(4,
 1)', '(5, 1)', '(6, 1)', '(7, 1)', '(8, 1)', '(8, 2)', '(8, 3)', '(8, 4)', '(8, 5)',
'(8, 6)', '(8, 7)', '(8, 8)', '(8, 9)', '(8, 10)', '(8, 11)', '(8, 12)', '(8, 13)', '(
8, 14)', '(8, 15)', '(8, 16)'])
vicfic@NAG3:~/stud/ai/ex2$
```

Learning Outcomes

- Implemented the A* algorithm using Python's heapq.
- Learn to properly represent grid search problems using recursive path structures and abstract base classes.
- Understood the importance of priority queues in efficiently managing the search frontier to guarantee optimal solutions